```
RRRRRRRRRRR      MMM          MMM     SSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSSSSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMMMMM    MMMMMM  SSS
RRR         RRR  MMM  MMM  MMM  MMM  SSS
RRR         RRR  MMM  MMM  MMM  MMM  SSS
RRR         RRR  MMM  MMM  MMM  MMM  SSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSS
RRRRRRRRRRRR     MMM          MMM     SSSSSSSSS
RRR   RRR        MMM          MMM             SSS
RRR   RRR        MMM          MMM             SSS
RRR   RRR        MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR      RRR     MMM          MMM             SSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
RRR         RRR  MMM          MMM  SSSSSSSSSSS
```

FILEID**RM3CMPRSS

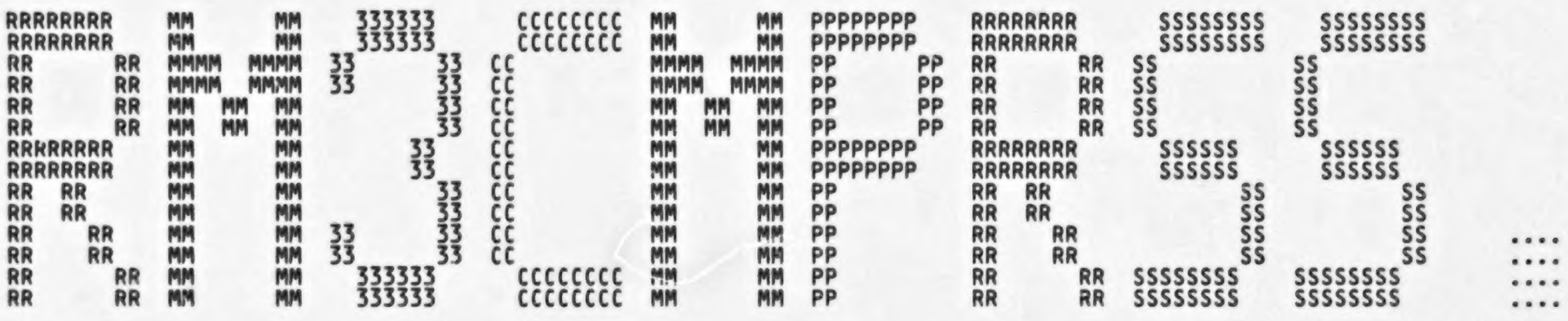

Table of contents

```
0000     1          $BEGIN  RM3CMPRSS,000,RM$RMS3,<>,<PIC,NOWRT,QUAD>
0000     2
0000     3  ;****************************************************************************
0000     4  ;*                                                                          *
0000     5  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     6  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     7  ;*  ALL RIGHTS RESERVED.                                                    *
0000     8  ;*                                                                          *
0000     9  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    10  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    11  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    12  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    13  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    14  ;*  TRANSFERRED.                                                            *
0000    15  ;*                                                                          *
0000    16  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    17  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    18  ;*  CORPORATION.                                                            *
0000    19  ;*                                                                          *
0000    20  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    21  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    22  ;*                                                                          *
0000    23  ;*                                                                          *
0000    24  ;****************************************************************************
0000    25  ;
0000    26
0000    27  ;++
0000    28  ;
0000    29  ; Facility:     RMS32 Index Sequential File Organization
0000    30  ;
0000    31  ; Abstract:
0000    32  ;
0000    33  ;       This modules contains the routines to handle compressed buckets
0000    34  ;       and compressed records.
0000    35  ;
0000    36  ; Environment:          VAX/VMS Operating System
0000    37  ;
0000    38  ; Author:       Todd M. Katz            Creation Date:  13-Aug-1982
0000    39  ;
0000    40  ; Modified By:
0000    41  ;
0000    42  ;       V03-008 TMK0006         Todd M. Katz            03-Feb-1983
0000    43  ;               Add support for Recovery Unit Journalling and RU ROLLBACK
0000    44  ;               Recovery of ISAM files. This involves a change to RM$SRCH_CMPR.
0000    45  ;               Check both for IRC$V_DELETED and IRC$V_RU_DELETED before setting
0000    46  ;               the IRB$V_DUPS_SEEN flag. Previously, just IRC$V_DELETED was
0000    47  ;               being checked.
0000    48  ;
0000    49  ;       V03-007 TMK0005         Todd M. Katz            16-Sep-1982
0000    50  ;               The field IRB$B_SRCHFLAGS has been changed to a word in size.
0000    51  ;               Fix all the references to it.
0000    52  ;
0000    53  ;               If a record is encountered with a key that is an exact duplicate
0000    54  ;               of the search key, then set the bit IRB$V_DUP_KEY regardless
0000    55  ;               of whether the record is or isn't marked deleted if RMS is
0000    56  ;               currently positioning for insertion.
0000    57  ;
```

```
0000     58 ;
0000     59 ;                Performance enhancement. RMS does not have to call
0000     60 ;                RM$GETNEXT_REC to position to the next record in the bucket.
0000     61 ;                If this is an index record, then the address of the next record
0000     62 ;                is REC_ADDR + current key size + 2 for compression overhead.
0000     63 ;                If this is anyother type of record, (primary data or SIDR) then
0000     64 ;                RMS knows that the record size field makes up the last two bytes
0000     65 ;                of the record overhead, and can use the quantity there + the
0000     66 ;                record overhead to position to the next record.
0000     67 ;
0000     68 ;                At the present time, RMS positions past deleted records even
0000     69 ;                when the search would otherwise be terminated because of the
0000     70 ;                key value of the current record, the search key value, and the
0000     71 ;                goal of the search. This is incorrect, and inconsistant with the
0000     72 ;                manner in which the rest of the searching is performed. It
0000     73 ;                creates problems during next record positioning which always
0000     74 ;                tries to first position to the current record before positioning
0000     75 ;                to the next record, and thus, could end up positioning past a
0000     76 ;                stream's internal current record because its marked deleted, and
0000     77 ;                therefore wrongly assume that the record had been completely
0000     78 ;                deleted from the file. The solution to this problem is to return
0000     79 ;                the record that the search terminates at regardless of whether
0000     80 ;                the record is or isn't marked deleted, and to let the upper
0000     81 ;                level routines decide what to do if the record is in fact marked
0000     82 ;                deleted.
0000     83 ;
0000     84 ;                At the present time, RM$SRCH_CMPR always starts its search with
0000     85 ;                the first record in the current bucket. This is unacceptable
0000     86 ;                because of the above made change - ie, searches may now
0000     87 ;                terminate with deleted records, and thus, may have to resume
0000     88 ;                positioning somewhere within the bucket in order to find a
0000     89 ;                non-deleted record. Fortunately, this change is easy to make
0000     90 ;                provided several assumptions hold:
0000     91 ;
0000     92 ;                    1. The goal of the search does not change between invocations
0000     93 ;                       of RM$SRCH_CMPR.
0000     94 ;
0000     95 ;                    2. The search key does not change between invocations of
0000     96 ;                       RM$SRCH_CMPR.
0000     97 ;
0000     98 ;                    2. The bucket being searched is kept locked between invocations
0000     99 ;                       of this routine.
0000    100 ;
0000    101 ;                    3. The keys are always in ascending order in the bucket, and the
0000    102 ;                       compression of these keys are always correct.
0000    103 ;
0000    104 ;                If these assumptions hold true, then it will always possible to
0000    105 ;                resume the search in the middle of a bucket, and return whether
0000    106 ;                the next record has a key value equal to (if the goal of the
0000    107 ;                search is EQ) or GT (if the goal of the search is GT or EQ) the
0000    108 ;                search key.
0000    109 ;
0000    110 ;        V03-006 KBT0159         Keith B. Thompson       21-Aug-1982
0000    111 ;                Reorganize psects
0000    112 ;
0000    113 ;        V03-005 TMK0004         Todd M. Katz            13-Aug-1982
0000    114 ;                Completely re-wrote the routine responsible for searching
                             compressed buckets, and the routine responsible for determining
```

```
0000    115 ;
0000    116 ;
0000    117 ;
0000    118 ;
0000    119 ;
0000    120 ;
0000    121 ;--
```

the amount of front compression of records.

Added support for prologue 3 SIDRs to both the compressed key bucket searching routine and the front compression determining routine.

```
0000    123          .SBTTL  DEFINITIONS
0000    124
0000    125
0000    126 ;
0000    127 ; Internal Structure Symbol Definitions
0000    128 ;
0000    129
0000    130          $BKTDEF
0000    131          $IRBDEF
0000    132          $IFBDEF
0000    133          $IRCDEF
0000    134          $IDXDEF
```

```
0000   136          .SBTTL  RM$SRCH_CMPR - Search a Compressed Index, SIDR, or Data Bucket
0000   137 ;+++
0000   138 ;
0000   139 ; FUNCTIONAL DESCRIPTION:
0000   140 ;
0000   141 ;       This routine performs an equal search or a greater-than search on a
0000   142 ;       primary data, SIDR, or index bucket with compressed key records using
0000   143 ;       the search key found in keybuffer 2. The search may start with the first
0000   144 ;       record in the bucket, or with a record somewhere in the middle of the
0000   145 ;       bucket. When the search is completed, REC_ADDR is positioned to the
0000   146 ;       record to be returned, and R0 contains the status of the search.
0000   147 ;
0000   148 ;       This routine makes some basic assumptions which can not be violated
0000   149 ;       without expecting totally unpredicatble search results.
0000   150 ;
0000   151 ;       1. It is assumed that the keys of the records in the bucket are strictly
0000   152 ;          in ascending order, and that they are always as fully compressed
0000   153 ;          as they can be for the position they occupy.
0000   154 ;
0000   155 ;       2. The two key compression bytes always follow whatever record overhead
0000   156 ;          is present in the record (if any), regardless of the bucket type. The
0000   157 ;          first key compression byte is always the number of bytes of key
0000   158 ;          present, and the second key compression byte is always the amount of
0000   159 ;          front compression of the key.
0000   160 ;
0000   161 ;       3. Record overhead is a fixed quantity for each record type.
0000   162 ;          Furthermore, if a record has record overhead associated with it, the
0000   163 ;          record's size minus the record overhead is always stored in the last
0000   164 ;          two bytes of record overhead.
0000   165 ;
0000   166 ;       4. Whenever RMS is positioning for insertion it performs a greater-than
0000   167 ;          search.
0000   168 ;
0000   169 ;       5. The decision to terminate a search is based on the goal of the search
0000   170 ;          and the outcome of the comparison between the key of the record being
0000   171 ;          returned and the search key. It is never based on anything else about
0000   172 ;          the record, for example, whether the record is marked deleted or not.
0000   173 ;
0000   174 ;       6. If this routine is called to resume a search within a bucket then:
0000   175 ;
0000   176 ;          a. The bucket has been locked between routine invocations.
0000   177 ;          b. IRAB[IRB$L_LST_NCMP] still points to the last record with a zero
0000   178 ;             front-compressed key.
0000   179 ;          c. The goal of all consecutive routine invocations is identical
0000   180 ;             (either EQ or GT).
0000   181 ;          d. The search key has not changed between routine invocations.
0000   182 ;
0000   183 ; CALLING SEQUENCE:
0000   184 ;
0000   185 ;       BSBW    RM$SRCH_CMPR
0000   186 ;
0000   187 ; INPUT PARAMETERS:
0000   188 ;
0000   189 ;       R1      - if 0, greater-than or equal search
0000   190 ;                 if 1, greater-than search
0000   191 ;
0000   192 ; IMPLICIT INPUT:
```

```
0000   193 ;
0000   194 ;       R5      - BKT_ADDR              - address of bucket
0000   195 ;                     BKT$W_FREESPACE   - offset to first free byte in bucket
0000   196 ;                     BKT$B_INDEXNO     - key of reference of bucket
0000   197 ;                     BKT$B_LEVEL       - level of bucket
0000   198 ;
0000   199 ;       R6      - REC_ADDR              - address of where to begin search
0000   200 ;
0000   201 ;       R7      - IDX_DFN               - address of index descriptor
0000   202 ;
0000   203 ;       R9      - IRAB                  - address of IRAB
0000   204 ;                     IRB$L_KEYBUF      - address of contigious keybuffers
0000   205 ;                     IRB$B_KEYSZ       - size of the search key
0000   206 ;                     IRB$V_LAST_GT     - if set, GT search result ocurred
0000   207 ;                     IRB$V_POSINSERT   - if set, positioning for insertion
0000   208 ;                     IRB$W_SRCHFLAGS   - search flags
0000   209 ;
0000   210 ;       R10     - IFAB                  - address of IFAB
0000   211 ;                     IFB$W_KBUFSZ      - size of each keybuffer
0000   212 ;
0000   213 ; OUTPUT PARAMETERS:
0000   214 ;       NONE
0000   215 ;
0000   216 ; IMPLICIT OUTPUT:
0000   217 ;
0000   218 ;       IRB$V_DUP_KEY   - if set, there is at least one data record in the file
0000   219 ;                         (deleted or otherwise) with a key identical to that of
0000   220 ;                         the search key
0000   221 ;       IRB$V_DUPS_SEEN - if set, there is at least one primary data record with
0000   222 ;                         a key identical to that of the search key.
0000   223 ;       IRB$V_LAST_GT   - if set, the result of this search was that the search
0000   224 ;                         key was less than the record positioned to.
0000   225 ;       IRB$L_LST_NCMP  - address of last key with no front compression
0000   226 ;       IRB$L_LST_REC   - address of last primary data record in duplicate chain
0000   227 ;       IRB$L_REC_COUNT - number of the record found
0000   228 ;       REC_ADDR        - address of record found
0000   229 ;
0000   230 ; ROUTINE VALUE:
0000   231 ;
0000   232 ;       R0:  -1, search key < record found
0000   233 ;             0, search key = record found
0000   234 ;             1, search key > all records in the bucket
0000   235 ;
0000   236 ; SIDE EFFECTS:
0000   237 ;
0000   238 ;       If positioning for insertion within a primary data bucket, and a record
0000   239 ;       with a key value duplicate of the key of the record to be inserted is
0000   240 ;       encountered, IRB$V_DUP_KEY is set, IRB$V_DUPS_SEEN is set (provided
0000   241 ;       the record is not marked deleted), and the address of the record is
0000   242 ;       placed in IRB$L_LST_REC. In fact at the conclusion of the search, this
0000   243 ;       same field will contain the address of the last such duplicate
0000   244 ;       encountered while REC_ADDR points to the record that follows it which
0000   245 ;       is where the new record will be inserted. Of course, if the bucket is a
0000   246 ;       SIDR bucket, then there can only be one instance of a record with a
0000   247 ;       given key value in a bucket.
0000   248 ;
0000   249 ;       Whenever the search key is greater that the key of all the records in
```

```
                     0000   250 ;       the bucket, then REC_ADDR is left positioned at the end of the bucket
                     0000   251 ;       when this status is returned. This is independent of bucket type.
                     0000   252 ;
                     0000   253 ;---
                     0000   254
                     0000   255 RM$SRCH_CMPR::
091E 8F    BB        0000   256         PUSHR   #^M<R1,R2,R3,R4,R8,R11> ; save the working registers
```

```
                                0004   258
                                0004   259 ;
                                0004   260 ; Register Usage:
                                0004   261 ;
                                0004   262 ; R0  - Result of the comparison between the search key and the "last" record.
                                0004   263 ;
                                0004   264 ; R1  - Set to the type of bucket for determining the amount of record overhead.
                                0004   265 ;     - Number of bytes of search key and record key to be compared.
                                0004   266 ;     - Scratch register.
                                0004   267 ;
                                0004   268 ; R2  - Offset in the search key to the byte where the comparison between the
                                0004   269 ;       search key and the key of the "current" record is to begin.
                                0004   270 ;
                                0004   271 ; R3  - Working register for CMPC3 and CMPC5.
                                0004   272 ;       Working register during next record positioning.
                                0004   273 ;
                                0004   274 ; R4  - Number of bytes of record overhead, not including key compression bytes.
                                0004   275 ;
                                0004   276 ; R5  - Address of the beginning of the bucket in memory.
                                0004   277 ;
                                0004   278 ; R6  - Address in memory of the current record in the bucket.
                                0004   279 ;
                                0004   280 ; R7  - Address of the index descriptor.
                                0004   281 ;
                                0004   282 ; R8  - Address in memory of the first free byte in the bucket. Effectively the
                                0004   283 ;       address of the end of the bucket.
                                0004   284 ;
                                0004   285 ; R9  - Address of the IRAB.
                                0004   286 ;
                                0004   287 ; R10 - Address of the IFAB.
                                0004   288 ;
                                0004   289 ; R11 - Address of keybuffer 2. Effectively the address of the search key.
                                0004   290 ;
                                0004   291
    58    04 A5    3C          0004   292         MOVZWL  BKT$W_FREESPACE(R5),R8  ; compute the address of the first free
       58    55    C0          0008   293         ADDL2   R5,R8                   ; byte in the bucket, and put it in R8
                                000B   294
          58    56    D1      000B   295         CMPL    R6,R8                  ; if the bucket is empty, return a GT
                03    1F      000E   296         BLSSU   1$                     ; status (primary data or SIDR buckets)
                  0136  31      0010   297         BRW     140$                   ; otherwise continue
                                0013   298
    51    0C A5    9A          0013   299 1$:     MOVZBL  BKT$B_LEVEL(R5),R1     ; if this is an index bucket, then as
                04    13      0017   300         BEQLU   5$                     ; index records do not contain any
                54    D4      0019   301         CLRL    R4                     ; overhead intialize R4 to 0, and skip
                2B    11      001B   302         BRB     15$                    ; call to determine record overhead
                                001D   303
             01 A5    95      001D   304 5$:     TSTB    BKT$B_INDEXNO(R5)      ; if this is a primary data bucket,
                03    13      0020   305         BEQLU   10$                    ; setup R1 with a 0, else it is a SIDR
          51    01    CE      0022   306         MNEGL   #1,R1                  ; bucket and a -1 is placed in R1
                                0025   307
             FFD8'    30      0025   308 10$:    BSBW    RM$REC_OVHD            ; determine the amount of overhead in
       54    50    D0          0028   309         MOVL    R0,R4                  ; each record and store it in R4
                                002B   310
 51    55    0E    C1          002B   311         ADDL3   #BKT$C_OVERHDSZ,R5,R1  ; get address of first record in bucket
                                002F   312
       56    51    D1          002F   313         CMPL    R1,R6                  ; if RMS is to start search with first
                14    13      0032   314         BEQLU   15$                    ; record, then go start search
```

```
                     0034   316
                     0034   317 ;
                     0034   318 ; RMS is resuming a search, and not starting with the first record in the
                     0034   319 ; bucket. The rules for resuming a search are as follows:
                     0034   320 ;
                     0034   321 ; 1. If the goal of the search is GT, then as the previous record must have
                     0034   322 ;    been GT the search key, so must the current record. Therefore the search
                     0034   323 ;    can immediately terminate with this status.
                     0034   324 ;
                     0034   325 ; 2. If the goal of the search is EQ, then if the number of bytes the current
                     0034   326 ;    record's key is front compression is equal to or exceeds the size of the
                     0034   327 ;    search key, then the current record and the search key must also be EQ.
                     0034   328 ;    Therefore, such a status can be immediately returned.
                     0034   329 ;
                     0034   330 ; 3. If the goal of the search is EQ, but the number of bytes the current
                     0034   331 ;    record's key is front compressed is less than the size of the search key,
                     0034   332 ;    then the current record's key must be greater than the search key, and
                     0034   333 ;    such a status maybe immediately returned.
                     0034   334 ;
                     0034   335
                  0A E1  0034   336          BBC     #IRB$V_LAST_GT,-          ; if the result of the last routine
               03 42 A9  0036   337                  IRB$W_SRCHFLAGS(R9),12$ ; invocation was LT, then so is the
                  009B 31  0039   338 11$:    BRW     90$                       ; result of this contigious invocation
                           003C   339
         00A6 C9  01 A644 91  003C   340 12$:    CMPB    1(R6)[R4],-               ; determine whether the key of the
                           0043   341                  IRB$B_KEYSZ(R9)           ; current record is equal to or
                     F4 1F 0043   342          BLSSU   11$                       ; greater than the search key and
                  00CF 31  0045   343 13$:    BRW     110$                      ; return the appropriate status
                           0048   344
                     0048   345 ;
                     0048   346 ; RMS is to start the search with the first record in the bucket.
                     0048   347 ;
                     0048   348
                     0048   349 15$:    CSB     #IRB$V_LAST_GT,-          ; if the search is starting with the
                     0048   350                  IRB$W_SRCHFLAGS(R9)       ; first record in the bucket then there
                     004D   351                                            ; is no previous context
   0098 C9  56 D0    004D   352          MOVL    R6,IRB$L_LST_NCMP(R9)     ; the first non-compressed record
                     0052   353
5B 00B4 CA  3C       0052   354          MOVZWL  IFB$W_KBUFSZ(R10),R11     ; compute the address of keybuffer 2
   5B   60 A9  C0    0057   355          ADDL2   IRB$L_KEYBUF(R9),R11      ; and place it in R11
                     005B   356
      0094 C9  D4    005B   357          CLRL    IRB$L_REC_COUNT(R9)       ; RMS is positioned to the first record
```

```
                                005F    359
                                005F    360 ;
                                005F    361 ; The only time it is ever necessary to compare the key of the current record
                                005F    362 ; with the search key is when the number of bytes the key of the current record
                                005F    363 ; is compressed is the same as the offset to the character in the search key
                                005F    364 ; which terminated key comparison the last time it was done. The comparison is
                                005F    365 ; now done to see whether this previous comparison terminating character (and
                                005F    366 ; implicitely the rest that follow it in the search key) is still greater then
                                005F    367 ; its opposite in the key of the new current record.
                                005F    368 ;
                                005F    369 ; The comparison starts in the search key with the character that had previously
                                005F    370 ; terminated such a comparison, and the number of bytes of key to be compared
                                005F    371 ; is the minimum of the number of bytes thus remaining in the search key and the
                                005F    372 ; number of bytes in the key of the current record.
                                005F    373 ;
                                005F    374 ; Note that this strategy guarentees that a comparison is always done between
                                005F    375 ; the search key and the key of the first record in the bucket.
                                005F    376 ;
                                005F    377
                          52    D4    005F    378          CLRL    R2                          ; initialize the search key offset to 0
                                      0061    379
                00A6 C9   51    9A    0061    380 20$:     MOVZBL  IRB$B_KEYSZ(R9),R1          ; compute the number of bytes in the
                      51    52    82    0066    381          SUBB2   R2,R1                       ; search key remaining to be compared
                                      0069    382
                6644  51    91    0069    383          CMPB    R1,(R6)[R4]                 ; use the minimum of the search key
                      04    1B    006D    384          BLEQU   30$                         ; bytes remaining and the current record
                6644  51    9A    006F    385          MOVZBL  (R6)[R4],R1                 ; key size as the key comparison size
                                      0073    386
6B42    02 A644   51    29    0073    387 30$:     CMPC3   R1,2(R6)[R4],(R11)[R2]      ; if the search key is equal to or less
                      65    13    007A    388          BEQLU   100$                        ; than the current record key process
                      59    1A    007C    389          BGTRU   90$                         ; accordingly, otherwise position to the
                50    01    9A    007E    390          MOVZBL  #1,R0                       ; next record in the bucket
                                      0081    391
                                      0081    392 ;
                                      0081    393 ; Position to the record which follows the current record in the bucket. Before
                                      0081    394 ; performing this positioning, save the address of the old current record if it
                                      0081    395 ; was zero front compressed.
                                      0081    396 ;
                                      0081    397
          52    53    5B    C3    0081    398 40$:     SUBL3   R11,R3,R2                    ; compute terminating search key offset
                                      0085    399
          01 A644   95    0085    400 50$:     TSTB    1(R6)[R4]                   ; if the key of the current record is
                      05    12    0089    401          BNEQU   55$                         ; 0 front compressed, save its address
    0098 C9   56    D0    008B    402          MOVL    R6,IRB$L_LST_NCMP(R9)       ; before positioning to the next record
                                      0090    403
            0C A5   95    0090    404 55$:     TSTB    BKT$B_LEVEL(R5)             ; if this is an index bucket then next
                      0A    13    0093    405          BEQL    60$                         ; record position equals the current
                53    66    9A    0095    406          MOVZBL  (R6),R3                     ; record position + current record key
          56    02 A643   9E    0098    407          MOVAB   2(R6)[R3],R6                ; size + two bytes for the key
                      0A    11    009D    408          BRB     62$                         ; compression overhead
                                      009F    409
                56    54    C0    009F    410 60$:     ADDL2   R4,R6                       ; otherwise, next record position equals
             53   FE A6    3C    00A2    411          MOVZWL  -2(R6),R3                   ; current record position + record
                56    53    C0    00A6    412          ADDL2   R3,R6                       ; overhead + record size
                                      00A9    413
            0094 C9   D6    00A9    414 62$:     INCL    IRB$L_REC_COUNT(R9)         ; increment the record counter
```

```
                                00AD   416 ;
                                00AD   417 ; There are a number of circumstances under which the result of the comparison
                                00AD   418 ; between the key of the new current record and the search key is known or can
                                00AD   419 ; be quickly determined without actually performing the comparison.
                                00AD   420 ;
                                00AD   421 ; 1. If RMS has positioned to the end of the bucket, or to a RRV record within
                                00AD   422 ;    a primary data bucket then the search is terminated with a GT status.
                                00AD   423 ;
                                00AD   424 ; 2. If the search key was found to be equal to the key of the last record, but
                                00AD   425 ;    the front compression of the key of the current record is less than the
                                00AD   426 ;    size of the search key, then the search key will be less than the key of
                                00AD   427 ;    new current record and it is processed as such.
                                00AD   428 ;
                                00AD   429 ; 3. If the search key was found to be equal to the key of the last record, and
                                00AD   430 ;    the front compression of the key of the new current record is either equal
                                00AD   431 ;    to or greater-than the size of the search key, then the search key will
                                00AD   432 ;    also be equal to the key of the new current record and is processed as
                                00AD   433 ;    such. The front compression of the key of the new current record maybe
                                00AD   434 ;    greater-than the size of the search key because RMS maybe performing a
                                00AD   435 ;    generic search with a search key smaller in size than the full size of a
                                00AD   436 ;    key for this key of reference.
                                00AD   437 ;
                                00AD   438 ; 4. If the search key was found to be greater-than the key of the last record,
                                00AD   439 ;    and the front compression of the key of the new current record is
                                00AD   440 ;    greater-than the position in the search key where the last comparison
                                00AD   441 ;    terminated, then the search key will also be greater-than the key of the
                                00AD   442 ;    new current record and RMS proceeds to position to the next record.
                                00AD   443 ;
                                00AD   444 ; 5. If the search key was found to be greater-than the key of the last record,
                                00AD   445 ;    but the front compression of the key of the new current record is less-than
                                00AD   446 ;    the position in the search key where the last comparison terminated, then
                                00AD   447 ;    the search key will be less-than the key of the new current record and is
                                00AD   448 ;    processed as such.
                                00AD   449 ;
                                00AD   450 ; In the remaining circumstances a direct comparison between the key of the new
                                00AD   451 ; current record and the search key is required, and is performed.
                                00AD   452 ;
                                00AD   453
     58    56    D1             00AD   454          CMPL    R6,R8                   ; if RMS is at the end of the bucket
           0C    1E             00B0   455          BGEQU   65$                     ; or has positioned ti a RRV record
        01 A5    89             00B2   456          BISB3   BKT$B_INDEXNO(R5),-      ; in a primary data bucket then
   51   0C A5                   00B5   457                  BKT$B_LEVEL(R5),R1       ; go return a status of GT (search key
           07    12             00B8   458          BNEQU   70$                     ; greater than all the records in the
   03 66   03    E1             00BA   459          BBC     #IRC$V_RRV,(R6),70$     ; bucket)
        0088     31             00BE   460 65$:     BRW     140$
                                00C1   461
           50    D5             00C1   462 70$:     TSTL    R0                      ; if the last comparison's result was GT
           09    14             00C3   463          BGTR    80$                     ; then go decide between cases 4 or 5 or
                                00C5   464                                          ; whether a key comparison must be made
                                00C5   465
52    01 A644    91             00C5   466          CMPB    1(R6)[R4],R2            ; if CASE 2 holds true process as
           0B    1F             00CA   467          BLSSU   90$                     ; less-than, but if CASE 3 holds true
           53    11             00CC   468          BRB     115$                    ; process as equal
                                00CE   469
52    01 A644    91             00CE   470 80$:     CMPB    1(R6)[R4],R2            ; if CASE 4 holds true go position to
           B0    1A             00D3   471          BGTRU   50$                     ; the next record, but if CASE 5 holds
           8A    13             00D5   472          BEQLU   20$                     ; true process as less-than otherwise
```

```
                                00D7   473
                                00D7   474 ;
                                00D7   475 ; RMS has positioned to a record whose key is greater than that of the search
                                00D7   476 ; key. Return this status.
                                00D7   477 ;
                                00D7   478
                50   01   CE    00D7   479 90$:    MNEGL   #1,R0                   ; setup the status in R0 to be LT and
                                00DA   480         SSB     #IRB$V_LAST_GT,-        ; save that the result of this search
                                00DA   481                 IRB$W_SRCHFLAGS(R9)     ; was GT in case the search must resume
                     6D   11    00DF   482         BRB     150$                    ; go return this status
                                00E1   483
                                00E1   484 ;
                                00E1   485 ; On an actual search key - current record key comparison, the parts of the
                                00E1   486 ; key that were compared were found to be equivalent. This does not necessairly
                                00E1   487 ; mean that the two keys are in fact identical. If the size of the search key
                                00E1   488 ; (including those characters front compressed but not rear-end truncated) is
                                00E1   489 ; less than or equal to the size of the key of the current record, then in fact
                                00E1   490 ; the two keys are identical, and are processed as such. However, if because of
                                00E1   491 ; rear-end truncation the search key is greater in size then the key of the
                                00E1   492 ; current record, then the comparison between the two keys must be continued.
                                00E1   493 ; This is done by extending the key of the current record by the last character
                                00E1   494 ; present, and comparing the remaining bytes in the search key with it alone. If
                                00E1   495 ; the two keys are still identical they are processed as such; otherwise, they
                                00E1   496 ; are processed depending on whether the search key is greater-than or
                                00E1   497 ; less-than the key of the current record.
                                00E1   498 ;
                                00E1   499
      51   01 A644    6644    81  00E1   500 100$:   ADDB3   (R6)[R4],1(R6)[R4],R1   ; if the size of the search key is
           51   00A6 C9         91  00E8   501         CMPB    IRB$B_KEYSZ(R9),R1      ; less-than or equal to the size of the
                     28         1B  00ED   502         BLEQU   110$                    ; current record's key, process as equal
                                00EF   503
           52   53    5B        C3  00EF   504         SUBL3   R11,R3,R2               ; determine where in the search key the
                      53        D4  00F3   505         CLRL    R3                      ; comparison stopped and how many search
      53   00A6 C9    52        83  00F5   506         SUBB3   R2,IRB$B_KEYSZ(R9),R3   ; key bytes remain to be compared
                                00FB   507
                51    6644      9A  00FB   508         MOVZBL  (R6)[R4],R1             ; compute the offset to the last
           51   01 A441         9E  00FF   509         MOVAB   1(R4)[R1],R1            ; character in the current record's key
                                0104   510
53    6641    6641    01        2D  0104   511         CMPC5   #1,(R6)[R1],(R6)[R1],-  ; compare the remaining search key bytes
                     6842       0108   512                 R3,(R11)[R2]            ; with the current record key's last
                       C8       1A  010D   513         BGTRU   90$                     ; character, and continue processing
                       06       13  010F   514         BEQLU   110$                    ; depending upon whether they are
                50     01       9A  0111   515         MOVZBL  #1,R0                   ; identical, the search key is less-than
                     FF6A       31  0114   516         BRW     40$                     ; the current record's key or vice versa
```

```
                     0117   518
                     0117   519 ;
                     0117   520 ; The search key has been found to be identical with the key of the current
                     0117   521 ; record.
                     0117   522 ;
                     0117   523 ; If the goal of the search is to find an equal match then RMS is done and
                     0117   524 ; should return such a status provided the record is not a primary data record
                     0117   525 ; marked deleted. In such an instance, RMS continues the search with the next
                     0117   526 ; primary data record in the bucket.
                     0117   527 ;
                     0117   528 ; If the goal of the search is to find a greater-than match, then RMS will also
                     0117   529 ; continue the search with the next record in the bucket. However, before
                     0117   530 ; continuing the search, if RMS is positioning for insertion within a data
                     0117   531 ; bucket, then as the key of the new record will be identical to the key of the
                     0117   532 ; current record, RMS saves the address of the current record as the last record
                     0117   533 ; seen in the data bucket with this key value. RMS will also indicate that a
                     0117   534 ; a record with a key duplicate to that of the new record has been seen by
                     0117   535 ; setting a bit in the IRAB, provided the current record is not marked deleted,
                     0117   536 ; and it will indicate that some record with this key value has been seen by
                     0117   537 ; setting another bit in the IRAB, regardless of the setting of the current
                     0117   538 ; record.
                     0117   539 ;
                     0117   540
            50   D4 0117   541 110$:   CLRL    R0                      ; setup the status in R0 to be equal
                     0119   542
            6E   D5 0119   543         TSTL    (SP)                    ; if the goal of the search is an equal
            31   13 011B   544         BEQLU   150$                    ; match then go an EQ status, otherwise
       52   53   5B C3 011D   545         SUBL3   R11,R3,R2               ; compute terminating search key offset
                     0121   546
          0C A5   95 0121   547 115$:   TSTB    BKT$B_LEVEL(R5)         ; if rms is not currently positioning
            20   12 0124   548         BNEQU   130$                    ; for insertion within a data bucket,
            00   E1 0126   549         BBC     #IRB$V_POSINSERT,-      ; then continue the search for a record
        1B 42 A9       0128   550                 IRB$W_SRCHFLAGS(R9),130$; with a key greater-than the search key
                     012B   551
                     012B   552         SSB     #IRB$V_DUP_KEY,-        ; otherwise, save the address of the
                     012B   553                 IRB$W_SRCHFLAGS(R9)     ; current record, set a bit indicating
        4C A9   56   D0 0130   554         MOVL    R6,IRB$L_LST_REC(R9)    ; that a duplicate key was encountered
          01 A5   95 0134   555         TSTB    BKT$B_INDEXNO(R5)       ; during the search, and indicate that
            08   12 0137   556         BNEQ    120$                    ; duplicates have been seen during the
        09 66   02 E0 0139   557         BBS     #IRC$V_DELETED,(R6),130$; search if the current record is a
           66   05 E0 013D   558         BBS     #IRC$V_RU_DELETE,(R6),- ; SIDR, or if the current record is a
            05   0140   559                 130$                    ; primary data record that is not
          80 8F   88 0141   560 120$:   BISB2   #IRB$M_DUPS_SEEN,-     ; marked either deleted or deleted
          44 A9       0144   561                 IRB$B_SPL_BITS(R9)      ; within a Recovery Unit
           FF3C   31 0146   562 130$:   BRW     50$
```

```
                     0149   564
                     0149   565 ;
                     0149   566 ; RMS has found that the search key is greater-than the key of every record
                     0149   567 ; in the bucket. In this case RMS will immediately terminate the search with
                     0149   568 ; a greater-than status.
                     0149   569 ;
                     0149   570
      50   01   9A   0149   571 140$:   MOVZBL  #1,R0                    ; go terminate the search with a status
           15   11   014C   572         BRB     160$                     ; of greater-than
                     014E   573
                     014E   574 ;
                     014E   575 ; Return the status of the search to the caller of this routine. If the bucket
                     014E   576 ; that was searched was a data level bucket, and RMS was not positioning for
                     014E   577 ; insertion, then save the address of the current record as the last zero
                     014E   578 ; front compressed record encountered provided it is zero front compressed
                     014E   579 ; and there is a record to be returned (ie - the status of the search is not
                     014E   580 ; greater-than).
                     014E   581 ;
                     014E   582
    0C A5   95       014E   583 150$:   TSTB    BKT$B_LEVEL(R5)          ; immediately return the appropriate
           10   12   0151   584         BNEQU   160$                     ; status if this is not a data bucket
                     0153   585
           00   E0   0153   586         BBS     #IRB$V_POSINSERT,-       ; if RMS is positioning for insertion
    0B 42 A9         0155   587                 IRB$W_SRCHFLAGS(R9),160$; then immediately return status
                     0158   588
     01 A644   95    0158   589         TSTB    1(R6)[R4]                ; if the current record is zero front
           05   12   015C   590         BNEQU   160$                     ; compressed then save its address as
0098 C9    56   D0   015E   591         MOVL    R6,IRB$L_LST_NCMP(R9)    ; the last seen zero-compressed record
                     0163   592
     091E 8F   BA    0163   593 160$:   POPR    #^M<R1,R2,R3,R4,R8,R11>  ; restore the registers used and
                05   0167   594         RSB                              ; return
```

```
                        .SBTTL  RM$FRNT_CMPR - Compute a Record's Front Compression Count
;+++
;
; FUNCTIONAL DESCRIPTION:
;
;       This routine's responsibility is to take a proposed point of insertion
;       of a new record, and determine the amount of front compression the key
;       of the new record will have if it is inserted there. The record maybe
;       a primary data, an index, or a SIDR record. There are two assumptions
;       which this routine makes:
;
;       1. The keys of the records in the bucket are in ascending order and are
;          correctly compressed (ie - they are as compressed as they can be for
;          their place in the bucket).
;
;       2. Each record in the bucket is preceeded by the same number of bytes of
;          overhead, a constant for the type of file and type of bucket, and
;          key compression overhead always consists of two bytes - the first the
;          size of the key that is present, and the second the number of bytes
;          of front compression.
;
; INPUT PARAMETERS:
;
;       R6      - address where new record is to be inserted
;       R8      - address of key of new record
;                 (including key compression overhead)
;
; IMPLICIT INPUT:
;
;       R5      - BKT_ADDR                  - address of primary/index/SIDR bucket
;                     BKT$B_INDEXNO         - index number of bucket
;                     BKT$B_LEVEL           - level of bucket
;
;       R7      - IDX_DFN                   - address of index descriptor
;                     IDX$B_KEYSZ           - size of key
;
;       R9      - IRAB                      - address of IRAB
;                     IRB$L_LST_NCMP        - address of last key not compressed
;                     IRB$L_REC_COUNT       - number of preceeding records
;
;       R10     - IFAB                      - address of IFAB
;
; OUTPUT PARAMETERS:
;       NONE
;
; IMPLICIT OUTPUT:
;       NONE
;
; ROUTINE VALUE:
;
;       R0      - number of characters which can be front compressed
;
; SIDE EFFECTS:
;       NONE
;
;---
```

```
                          0168   653
                          0168   654 RM$FRNT_CMPR::
                 081E 8F  BB 0168   655         PUSHR   #^M<R1,R2,R3,R4,R11>    ; save the working registers
                 0094 C9  DD 016C   656         PUSHL   IRB$L_REC_COUNT(R9)     ; save the record count
                      7E  D4 0170   657         CLRL    -(SP)                   ; 0 is current front compression guess
                          0172   658
                          0172   659 ;
                          0172   660 ; If the size of the key is zero bytes, or if the new record is to be inserted
                          0172   661 ; at the beginning of the bucket, then go return indicating that the key of the
                          0172   662 ; new record will not have to be front compressed.
                          0172   663 ;
                          0172   664
                      68  95 0172   665         TSTB    (R8)                    ; if the new record's key size is zero
                      5D  13 0174   666         BEQLU   50$                     ; then return 0 bytes front compresion
                          0176   667
               51 55  0E  C1 0176   668         ADDL3   #BKT$C_OVERHDSZ,R5,R1   ; if the new record is to be inserted as
                  51  56  D1 017A   669         CMPL    R6,R1                   ; the first record in the bucket then
                      54  1B 017D   670         BLEQU   50$                     ; go return 0 bytes front compression
```

```
                     017F    672
                     017F    673 ;
                     017F    674 ; Before a determination can be made of the front compression that will be
                     017F    675 ; required for the key of the new record there are some necessary preparations.
                     017F    676 ;
                     017F    677 ;
                     017F    678 ; Register Usage:
                     017F    679 ;
                     017F    680 ; R0  - Size of the key of the current record in the bucket.
                     017F    681 ;
                     017F    682 ; R1  - Set to the type of bucket for determining the amount of record overhead.
                     017F    683 ;       Offset to the last character of the current record's key.
                     017F    684 ;
                     017F    685 ; R2  - Offset to the character in the key of the new record where the
                     017F    686 ;       comparison is to resume.
                     017F    687 ;
                     017F    688 ; R3  - Number of bytes of the new record's key remaining to be compared with
                     017F    689 ;       the key of the current record.
                     017F    690 ;
                     017F    691 ; R4  - Number of bytes of record overhead, not including key compression bytes.
                     017F    692 ;
                     017F    693 ; R5  - Address of the beginning of the bucket in memory.
                     017F    694 ;
                     017F    695 ; R6  - Address in memory of the current record in the bucket.
                     017F    696 ;
                     017F    697 ; R7  - Addre s of the index descriptor.
                     017F    698 ;
                     017F    699 ; R8  - Address of the key of the new record to be inserted.
                     017F    700 ;
                     017F    701 ; R9  - Address of the IRAB.
                     017F    702 ;
                     017F    703 ; R10 - Address of the IFAB.
                     017F    704 ;
                     017F    705 ; R11 - Address in memory of the bucket address where the new record is to be
                     017F    706 ;       inserted.
                     017F    707 ;
                     017F    708
     5B    56    D0  017F    709          MOVL    R6,R11                      ; save the point of insertion in R11 and
56   0098  C9    D0  0182    710          MOVL    IRB$L_LST_NCMP(R9),R6       ; initialize REC_ADDR to the address of
                     0187    711                                              ; the last zero-compressed record
                     0187    712
 51    0C   A5    9A  0187    713          MOVZBL  BKT$B_LEVEL(R5),R1          ; if this is an index bucket, then as
            04    13  018B    714          BEQLU   10$                         ; index records do not contain any
            54    D4  018D    715          CLRL    R4                          ; overhead initialize R4 to 0, and skip
            0E    11  018F    716          BRB     30$                         ; call to determine record overhead
                     0191    717
          01   A5    95  0191    718 10$:    TSTB    BKT$B_INDEXNO(R5)           ; if this is a primary data bucket,
            03    13  0194    719          BEQL    20$                         ; setup R1 with a 0, else it is a $IDR
       51   01    CE  0196    720          MNEGL   #1,R1                       ; bucket and a -1 is placed in R1
                     0199    721
             FE64'   30  0199    722 20$:    BSBW    RM$REC_OVHD                 ; determine the amount of overhead in
       54    50    D0  019C    723          MOVL    R0,R4                       ; each record and store it in R4
```

```
                                019F    725
                                019F    726 ;
                                019F    727 ; The records in the bucket are assumed to be in ascending order and correctly
                                019F    728 ; compressed. Therefore, if RMS's current best guess for the front compression
                                019F    729 ; of the key of the new record is less then the front compression count of the
                                019F    730 ; key of the current record, then there will be no need to compare the two keys.
                                019F    731 ; because the current record's key can not contribute any more to the
                                019F    732 ; compression of the key of the new record then was contributed by the key of
                                019F    733 ; last record the new record's key was compared with. Only if the current front
                                019F    734 ; compression estimate and the front compression count of the current record are
                                019F    735 ; the same will it be necessary to compare the two keys, because only then can
                                019F    736 ; the key of the current record influence the compression of the key of the new
                                019F    737 ; record.
                                019F    738 ;
                                019F    739
01 A644    6E    91             019F    740 30$:    CMPB    (SP),1(R6)[R4]          ; if compression counts arn't identical
                  25    12      01A4    741         BNEQ    40$                     ; then go position to the next record
                                01A6    742
                                01A6    743 ;
                                01A6    744 ; Compare the key of the new record with the key of the current record. Because
                                01A6    745 ; the current record's key is fully compressed, rear-end truncated as well as
                                01A6    746 ; front compressed, it will be necessary to extend it by its last character as
                                01A6    747 ; necessary. Furthermore, the comparison starts in the key of the new record,
                                01A6    748 ; not with its first character, but with the first character past those RMS has
                                01A6    749 ; already determined will be front compressed.
                                01A6    750 ;
                                01A6    751
   50    6644    9A             01A6    752         MOVZBL  (R6)[R4],R0             ; setup R0 and R1 with the size of and
51    01 A044    9E             01AA    753         MOVAB   1(R0)[R4],R1            ; offset to the last character in the
                                01AF    754                                         ; current record's key respectively
                                01AF    755
      52    6E    D0            01AF    756         MOVL    (SP),R2                 ; setup R2 and R3 with the offset to
         53   20 A7    9A       01B2    757         MOVZBL  IDX$B_KEYSZ(R7),R3      ; the first character to be compared
              53    52    C2    01B6    758         SUBL2   R2,R3                   ; and the number of bytes to compare in
                                01B9    759                                         ; the new record's key respectively
                                01B9    760
                                01B9    761
53   6641    02 A644    50    2D 01B9    762         CMPC5   R0,2(R6)[R4],(R6)[R1],- ; compare the key of the new record
                02 A842          01C1    763                 R3,2(R8)[R2]        ; with the key of the current record
                                01C4    764
            6E    53    58    C3 01C4    765         SUBL3   R8,R3,(SP)              ; compute a new best guess for the front
                  6E    02    C2 01C8    766         SUBL2   #2,(SP)                 ; compression of the new record's key
                                01CB    767                                         ; correcting for compression overhead
```

```
                     01CB    769
                     01CB    770 ;
                     01CB    771 ; Increment the current record pointer to the next record in the bucket. If the
                     01CB    772 ; address of the new current record is the same as the address of the point
                     01CB    773 ; of insertion of the new record, then go return the number of bytes the key of
                     01CB    774 ; the new record will have to be front compressed. Otherwise, go determine
                     01CB    775 ; whether the front compression of the key of the current record is the same
                     01CB    776 ; as RMS's current guess of the front compression of the key of the new record,
                     01CB    777 ; and the two keys will have to be compared, or whether the latter is
                     01CB    778 ; greater-than the former and they will not have to be compared.
                     01CB    779 ;
                     01CB    780
        FE32'   30   01CB    781 40$:    BSBW    RM$GETNEXT_REC          ; position to next record in the bucket
                     01CE    782
5B      56    D1     01CE    783         CMPL    R6,R11                  ; if RMS has positioned to the point of
        CC    1F     01D1    784         BLSSU   30$                     ; insertion then return, else continue
                     01D3    785
                     01D3    786 ;
                     01D3    787 ; Return the number of bytes the the key of the new record will have to be front
                     01D3    788 ; compressed if the new record is to go at the indicated place of insertion.
                     01D3    789 ;
                     01D3    790
        50  8ED0     01D3    791 50$:    POPL    R0                      ; load front compression count into R0
0094    C9  8ED0     01D6    792         POPL    IRB$L_REC_COUNT(R9)     ; restore the record count
081E    8F    BA     01DB    793         POPR    #^M<R1,R2,R3,R4,R11>    ; restore the working registers and
              05     01DF    794         RSB                             ; return
                     01E0    795         .END
```

```
$$.PSECT_EP                     = 00000000
$$RMSTEST                       = 0000001A
$$RMS_PBUGCHK                   = 00000010
$$RMS_TBUGCHK                   = 00000008
$$RMS_UMODE                     = 00000004
BKT$B_INDEXNO                   = 00000001
BKT$B_LEVEL                     = 0000000C
BKT$C_OVERHDSZ                  = 0000000E
BKT$W_FREESPACE                 = 00000004
IDX$B_KEYSZ                     = 00000020
IFB$W_KBUFSZ                    = 000000B4
IRB$B_KEYSZ                     = 000000A6
IRB$B_SPL_BITS                  = 00000044
IRB$L_KEYBUF                    = 00000060
IRB$L_LST_NCMP                  = 00000098
IRB$L_LST_REC                   = 0000004C
IRB$L_REC_COUNT                 = 00000094
IRB$M_DUPS_SEEN                 = 00000080
IRB$V_DUP_KEY                   = 00000008
IRB$V_LAST_GT                   = 0000000A
IRB$V_POSINSERT                 = 00000000
IRB$W_SRCHFLAGS                 = 00000042
IRC$V_DELETED                   = 00000002
IRC$V_RRV                       = 00000003
IRC$V_RU_DELETE                 = 00000005
RM$FRNT_CMPR                      00000168 RG    01
RM$GETNEXT_REC                    ********   X   01
RM$REC_OVHD                       ********   X   01
RM$SRCH_CMPR                      00000000 RG    01
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
RM$RMS3	000001E0 (480.)	01 (1.)	PIC USR CON REL GBL NOSHR EXE RD NOWRT NOVEC QUAD
ABS	00000000 (0.)	02 (2.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	31	00:00:00.07	00:00:01.14
Command processing	111	00:00:00.78	00:00:04.80
Pass 1	240	00:00:05.98	00:00:19.78
Symbol table sort	0	00:00:00.75	00:00:01.49
Pass 2	164	00:00:02.06	00:00:06.21
Symbol table output	5	00:00:00.05	00:00:00.13
Psect synopsis output	1	00:00:00.02	00:00:00.34
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	554	00:00:09.71	00:00:33.89

The working set limit was 1350 pages.

34246 bytes (67 pages) of virtual memory were used to buffer the intermediate code.
There were 30 pages of symbol table space allocated to hold 509 non-local and 34 local symbols.
795 source lines were read in Pass 1, producing 14 object records in Pass 2.
16 pages of virtual memory were used to define 15 macros.

+----------------------------+
! Macro library statistics !
+----------------------------+

Macro library name	Macros defined
_255DUA28:[RMS.OBJ]RMS.MLB;1	8
_255DUA28:[SYS.OBJ]LIB.MLB;1	0
_255DUA28:[SYSLIB]STARLET.MLB;2	3
TOTALS (all libraries)	11

597 GETS were required to define 11 macros.

There were no errors, warnings or information messages.

```
MACRO/LIS=LIS$:RM3CMPRSS/OBJ=OBJ$:RM3CMPRSS MSRC$:RM3CMPRSS/UPDATE=(ENH$:RM3CMPRSS)+EXECML$/LIB+LIB$:RMS/LIB
```

RM2UPDDEL
LIS

RM3CLOSE
LIS

RM3ALLBKT
LIS

RM2CREATE
LIS

RM2GET
LIS

RM2PUT
LIS

RM3BKTIO
LIS

RM3BKTSPL
LIS

RM3CMPKEY
LIS

RM3CMPRSS
LIS

RM2EXTEND
LIS

RM2FMTBKT
LIS

RM3BUG
LIS

RM2OPEN
LIS

RM3FACE
LIS

RM3DISCON
LIS

RM3CONN
LIS

RM3DELETE
LIS

RM3CREATE
LIS